05. Funciones

Programación - 1º DAM

Luis del Moral Martínez

versión 20.10

Contenidos del tema

1. Funciones
2. Ámbito de variables y funciones
3. Compilación separada

1. Funciones

Concepto de función

- C++ se puede usar como lenguaje de **programación estructurada**
- De esta forma, las funciones permiten subdividir el código en **unidades funcionales**
- Una **función** es un módulo que realiza una tarea concreta (o parte del problema)
- Una **función** no debería tener más de **10-15 líneas** de código
- Beneficios de utilizar funciones:
  - Permiten aislar mejor el problema
  - Se escriben programas correctos de forma más rápida
  - Permiten escribir programas que son más fáciles de mantener y actualizar

1. Funciones

Estructura de una función (1)

- Una función está formada por un **conjunto de sentencias**
- La función se pude llamar desde cualquier parte del programa
- Las funciones no se pueden anidar
- La estructura de una función es la siguiente:

```
tipo_valor_devuelto nombreFuncion (parámetros)
{
    sentencias;
    return expresión;
}
```

Cuerpo de la función (entre llaves)

1. Funciones

Estructura de una función (2)

- **Tipo de valor devuelto**: tipo de valor que devuelve la función (**int**, **double**…)
  - Si no devuelve nada, el tipo es **void**
- **Nombre de la función**: identificador o nombre de la función
  - Se siguen los mismos criterios que para los nombres de las variables
- **Lista de parámetros**: lista de variables que recibe la función como parámetro separadas por **,**
- **Sentencia return**: permite devolver un valor (coincide con el tipo de valor devuelto)
- El **cuerpo de la función** (sentencias) se encuentra dentro de un bloque delimitado por **{** y **}**

1. Funciones

Estructura de una función (3)

- Es obligatorio especificar el **tipo de dato** de cada parámetros que recibe la función
  - Ejemplo: **int** parametro1, **double** parametro2…
- Después de la llave de cierre de la función no se escribe punto y coma
- Los parámetros se pueden pasar por valor o por referencia (se estudiará en un tema posterior)
- No se pueden declarar funciones anidadas
- Las variables que se declaran dentro de una función son **locales** y no existen fuera de esta
- Abre el fichero **05_01_funciones_1.cpp**

1. Funciones

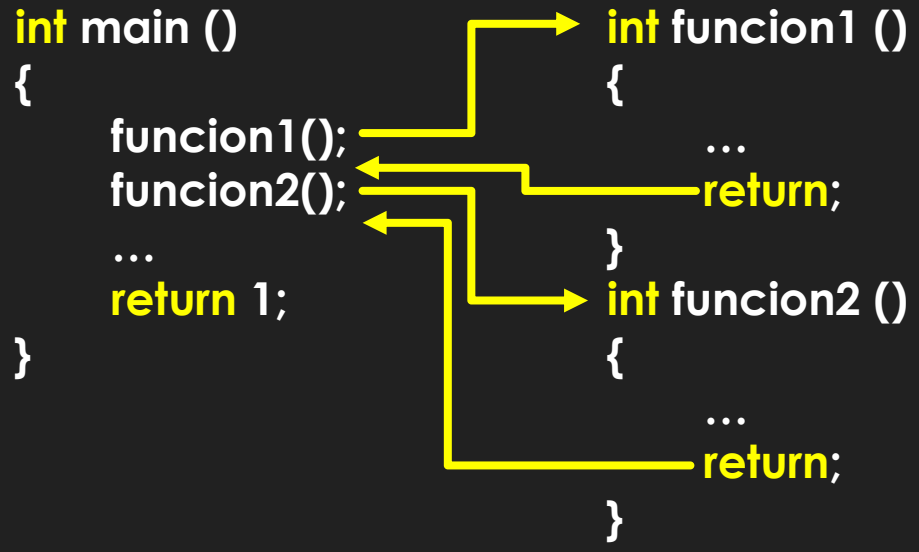
Estructura de una función (4)

- **Nombre de la función**
  - Debe comenzar con una letra o con un subrayado (_)
  - Puede contener letras, números o subrayados, con la longitud que queramos
  - Se distingue entre mayúsculas y minúsculas
- **Tipo de valor devuelto**
  - Si la función no devuelve nada (void), no se utilizará la sentencia return
  - Una función puede devolver un único valor (o una variable o un array, como estudiaremos después)
  - Si no usamos return debidamente, se podrá devolver un valor inesperado

1. Funciones

Llamada a una función

- Para que se ejecute el código de la función, debemos **llamar** o **invocar** a la función
- Podemos llamar a una función desde la función **main**() o desde cualquier otra función



- Abre el fichero **05_02_funciones_2.cpp**

1. Funciones

Declaración de funciones (1)

- C++ requiere que una función sea declarada antes de ser utilizada
- Existen dos formas de hacerlo:
  1. Escribir el **prototipo** de la función antes de usarla
     - Después escribiremos el cuerpo de la función más abajo
  2. Escribir directamente el **cuerpo de la función** antes de usarla

1. Funciones

Declaración de funciones (2)

- **Método 1**: usar prototipo de funciones
  - Los prototipos solo incluyen el **tipo**, el **nombre** y la **lista de argumentos**
  - En la lista de argumentos podemos poner únicamente los **tipos de datos**, separados por **coma**
  - Después de la función main podemos escribir el **cuerpo de la función**, incluyendo sentencias y **return**

```
int funcion1 ();

int main ()
{
        funcion1();
        …
        return 1;
}
```

Ejemplos de prototipos

```
int sumar (int, int);
double raizCuadrada (double);

…
```

1. Funciones

Declaración de funciones (3)

- **Método 2**: escribir todo el cuerpo de la función antes de usarla
  - En este caso no hace falta escribir el prototipo de la función, puesto que ya la hemos declarado

```
int funcion1 ()
{
        …
        return 1;
}
int main ()
{
        funcion1();
        return 1;
}
```

- Abre el fichero **05_03_funciones_3.cpp**

1. Funciones

Declaración de funciones (4)

- **Parámetros constantes**:
  - Con el especificador **const** podemos indicar que un parámetro sea constante en la función
  - Los parámetros y variables constantes no pueden cambiar su valor

```
int funcion1 (const int numero)
{
        …
        return 1;
}
```

1. Funciones

Declaración de funciones (5)

- **Valores por omisión**
  - Es posible indicar un **valor por defecto** para un parámetro
  - El valor por defecto se usará si no hemos indicado ningún valor a la hora de llamar a la función

```
int funcion1 (int numero = 3)
{
        …
        return 1;
}
```

2. Ámbito de variables y funciones

Ámbito (alcance)

- El ámbito de una variable determina si es visible o no en una determinada función
- Si la variable es visible en una función, entonces podrá usarse en la misma
- Existen diferentes ámbitos:
  1. Ámbito del programa
  2. Ámbito del archivo fuente
  3. Ámbito de una función
  4. Ámbito de bloque
  5. Variables locales

2. Ámbito de variables y funciones

Ámbito del programa

- Las variables que tienen ámbito del programa se pueden usar en cualquier función
- Son **variables globales**
- Se declaran al principio del programa, fuera de cualquier función
- Su uso no está recomendado

```
int variable = 5;
int main ()
{
        …
        return 1;
}
```

2. Ámbito de variables y funciones

Ámbito del archivo fuente

- Únicamente tienen visibilidad en el archivo fuente en la que se declaran
- Son **variables globales pero únicamente dentro del fichero fuente**
- Se declaran al principio del programa, fuera de cualquier función, usando el modificador static
- Su uso no está recomendado

```
int static variable = 5;
int main ()
{
        …
        return 1;
}
```

2. Ámbito de variables y funciones

Ámbito de una función

- Este tipo de variables solo tienen visibilidad en el ámbito de la función
- Se pueden declarar en cualquier parte de la función (son locales a la función)

```
int funcion ()
{
        int variable = 5;
}

int main ()
{
        …
        return 1;
}
```

2. Ámbito de variables y funciones

Ámbito de bloque

- Estas variables se definen dentro de un bloque de código (delimitado por { y })
- Son locales a dicho bloque de código (fuera del bloque no existen)

```
if (numero > 3)
{
        int i;
        for (i = 0; i < 5; i++);
                funcion(i);
}
```

2. Ámbito de variables y funciones

Ámbito de bloque

- Estas variables se definen dentro de un bloque de código (delimitado por { y })
- Son locales a dicho bloque de código (fuera del bloque no existen)

```
if (numero > 3)
{
        int i;
        for (i = 0; i < 5; i++);
                funcion(i);
}
```

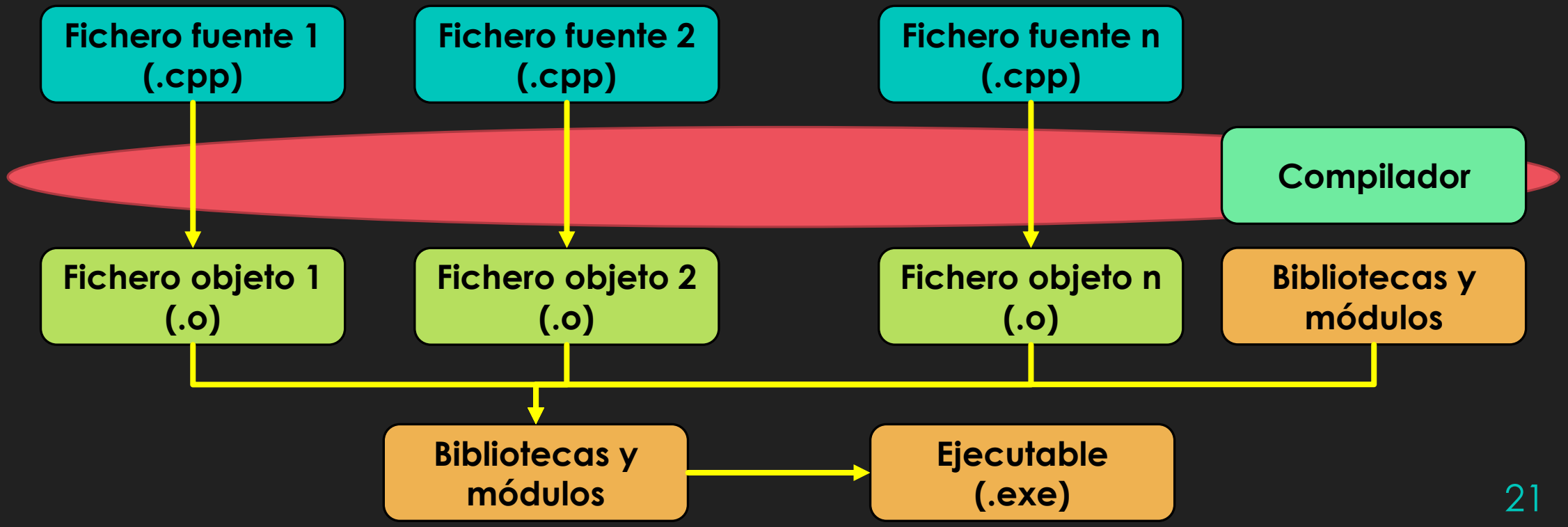
3. Compilación separada

Separar un programa en varios ficheros fuente

- Es posible separar un programa en varios ficheros fuente
- Si dividimos un programa, será más fácil de gestionar y mantener
- Todos los módulos se compilan por separado y el **enlazador** (**linker**) los enlaza
- Esto hace que los cambios en un programa sean más fáciles de realizar
- La recompilación es más rápida y sencilla (solo se recompilan los archivos que cambian)

3. Compilación separada

Proceso de compilación separada



Créditos de las imágenes y figuras

Cliparts e iconos

- **Obtenidos mediante la herramienta web IconFinder** (según sus disposiciones):
  - Diapositiva 1
  - Según la plataforma IconFinder, dicho material puede usarse libremente (free comercial use)
  - A fecha de edición de este material, todos los cliparts son free for comercial use (sin restricciones)

Resto de diagramas y gráficas

- Se han desarrollado en PowerPoint y se han incrustado en esta presentación
- Todos estos materiales se han desarrollado por el autor
  - Si se ha empleado algún icono externo, este se rige según lo expresado anteriormente